TVPP: A Research Oriented P2P Live Streaming System

João F. A. e Oliveira[1], Rodrigo S. M. Viana[1], Alex B. Vieira[2], Marcus V. M. Rocha[3], Sérgio V. A. Campos[1]

[1]Departamento de Ciência da Computação – UFMG

[2]Departamento de Ciência da Computação – UFJF

[3]Assembléia Legislativa de Minas Gerais

Abstract. *In the past five to six years, peer to peer live streaming has grown to support millions of users but the behavior of such systems is still not fully understood. Researchers experiment and try to understand popular/commercial platforms with proprietary code by crawling the network with one or more nodes, through network traffic analysis, reverse engineering and by recreating partial network graphs. However, most existing systems do not provide tools for analyzing the behavior in a comprehensive way, and these experiments cannot fully explain how this technology works. We introduce a tool called TVPP that supports peer-to-peer live streaming, and is being developed with the purpose of answering questions about how the underlying technology behaves and how it can be improved by allowing fine tuning of system parameters such as neighborhood size and bandwidth limits, and tests over alternative algorithms implementation. Further, it allows experimental data acquisition without additional network traffic analyzers which can impact experimental results.*

1. Introduction

Nowadays, P2P live streaming systems are widely available and extensively used by millions of users across many different architectures. An increase in the number of publications, in the past five years that discuss performance issues, load behavior, cross-platform analysis and many more topics related to P2P live streaming systems, shows that researchers are still trying to better understand the details behind these systems.

A major issue remains which makes it hard to study real life systems: most popular live systems, such as TvAnts[1], UUSee[2], SopCast[3], PPLive[4] and PPStream[5] are commercial applications, with no publicly available source code, which makes it harder for researchers to gather useful data or log files. Most studies targeting these systems deal with a black box and must rely on educated guesses with regard to system architecture, protocols and internals. Crawling the network, analyzing traffic, recreating partial network graphs are examples of methods frequently used to infer system structure and behavior [Vieira et al. 2009, Horvath et al. 2008, Silverston et al. 2009, Ali et al. 2006, Tang et al. 2009]. Moreover, in such a scenario where systems do not facilitate third party testing or reverse engineering, control of key protocol parameters, such as partnership or bandwidth limits, media buffer size, chunk scheduling and partner maintenance strategies, which would allow deeper experimentation and analysis, are absent.

[1]tvants.en.softronic.com;[2]www.uusee.com;[3]www.sopcast.org;[4]www.synacat.com;[5]www.ppstream.com

In this paper we present a new P2P live streaming system: TVPP. TVPP is designed to provide a similar service to popular proprietary commercial systems, such as SopCast, but with a few advantages for researchers.

- easy data acquisition: no need for additional network traffic analyzers
- configurability: several key parameters to experiment with
- modularity: easier testing of new algorithms
- collection of arbitrary data: one can change code to output what he needs
- exact analysis: one can look at the code to understand results

Through those advantages TVPP allows answering key questions, such as:

- what if there is only a limited number of partners for each peer?
- what if *x*% of peers are not willing or able to contribute with upstream bandwidth?
- what if downstream rate is reduced for some time?
- what if *x*% of some peers are receiving contents A and sending contents B (e.g. contents B could be contents A with added user comments)?

TVPP focuses on addressing the questions mentioned before and many other key design aspects of P2P live streaming systems. Also, TVPP source code is available for experimentation and modification. Through this paper we present the TVPP as a tool describing its design choices, structure, and what can be done with it. We have already conducted tests configuring TVPP with parameters to mimic SopCast behavior. The results show that, in this case, our system has performance comparable to that of SopCast.

The remainder of this paper is organized as follows. Several design choices for TVPP are mentioned in Section 2. After that we describe a quick "how to use" TVPP, detailing a few parameters available. In the next section we discuss related work. Section 5 expresses our conclusions, remarks and future work. And, finally, Section 6 details the proposed demonstration that will show our tool features.

2. The TVPP Design

TVPP is a mesh-pull P2P live streaming system, using the structures similar to those of the most popular platforms [Sentinelli et al. 2007]. An interesting feature is that TVPP has been designed to be expandable, as described below, since it aims to experiment with a wide range of questions, and therefore must be able to include additional monitoring mechanisms. Some of the key mechanisms that have been developed to maintain this functionality are particularly interesting to explore in further detail, such as chunk scheduling, overlay maintenance and logging. In Section 3 we explores a full list of configurable parameters present on TVPP.

2.1. Overlay Maintenance

In mesh-pull applications, peers are organized in a mesh like network, without any hierarchy, where each peer only requests or sends data to its partners. This design has the advantage of being scalable and fault resilient [Fodor and Dan 2007, Hei et al. 2008].

In TVPP, as in SopCast, mesh construction and maintenance relies on a centralized bootstrap server. For each channel available in the system, the bootstrap stores a list of nodes connected to that channel, the channel's source peer and its most recent produced chunk ID. In order to keep each channel's peer list up to date, peers send a *ping* packet to

the bootstrap periodically. The channel source, or server peer, also sends ping messages, which also updates the last generated chunk ID value for that channel. This value is used in a further moment to initialize new peers. Peers are removed from their channels' peer lists if they fail to report their existence to the bootstrap for a configurable period, typically a few seconds.

A channel's source peer (one that creates media and starts its distribution) announces its intent to create a channel to the bootstrap. Other peers join the network sending a peer request message to the bootstrap asking for peers watching an existing channel. After receiving a peer request message, the bootstrap selects a subset of peers from that channel's peers list and sends it back to the requester. This message also contains the last generated chunk ID so new peers know from which chunk to start asking.

Upon receiving the requested subset from the bootstrap, peers store them in a candidate peer list. Until it reaches neighborhood size limit, they will periodically try to establish a partnership with some candidates, turning that candidate into a partner. The candidate selection strategy is configurable and new strategies can be implemented. Partners may face connection issues or leave the network. In these cases they can be turned back into candidates or be dropped. Also, from time to time, peers will send new peer request messages to the bootstrap in order to refresh their candidates list. Another feature is that, periodically, partnerships can be undone. The selection strategy of partnerships to be undone is also configurable.

2.2. Chunk Scheduling

Another technique implemented in TVPP is described in [Zhang et al. 2005], where authors propose a model – known as the data-driven model. According to this model, all peers would only request data to neighbors that actually have the data. This is possible by making each peer broadcast periodically which chunks of data they have. This design generates more control overhead, but it prevents request/transmission redundancy.

The broadcasted message contains the buffer map which has an integer indicating the newest chunk of media present on the peers' buffer and a bit map where each position determines the presence or not of a previous chunk. The nth position of the map represents the id of the newest chunk minus n. Thus each buffer map covers a dynamic range of chunk IDs. Buffer map messages are also used as an "I am alive" message. If a peer stays more than a configurable period of time without receiving it from a neighbor, the peer removes that neighbor from its list.

The media server peer will naturally be the first node to have any chunk available. Through buffer map messages its partners will know that it has some chunks that they do not have. This behavior is reproduced over the entire network, once any node announces the presence of a chunk its partners might try to request, if they do not have it. Every few milliseconds each peer compares its own buffer map with its partners' maps looking for the newest chunk to request. Using a configurable selection strategy, it then selects a partner to serve that chunk and adds it to a finite request list signaling its intent to ask that partner for that particular chunk. The oldest chunk request is dropped if a new request arrives and the list is full. This request scheduling rule follows the earliest deadline first (EDF) rule, in which chunks closer to meet the deadline are requested first. After receiving a chunk, the respective request is removed from the request list, the chunk is

stored and the buffer map sets it as present.

2.3. Logging

Ping messages that are sent to the bootstrap also carry performance data. Using the bootstrap as the logging server increases the traffic at the bootstrap, but it has the advantage that there is no need for a special server to store log data.

These special ping – or *log* messages – include the following performance data: the number of packets generated, sent and received per second, packets missed since the last log message (a discontinuity measure), average hop count for each chunk received, and neighborhood size. The list can be easily extended, as new measures are envisioned and required. A more detailed chunk performance data is sent also in each message. This chunk data relates to one sample chunk sent during this period and is composed by its chunk ID, hop count and timestamps indicating when it was generated or consumed. All that data are used to get useful insights about latency, deadline miss rate, and distance that a chunk travels before being delivered. These are especially interesting because they are very hard (in some cases impossible) to be captured from commercial systems.

Another feature is that each peer periodically sends to the bootstrap a list of what other peers they are connected to. This is done through a different message with a list of peer addresses and a timestamp upon bootstrap receival. With those reports, one may track the evolution of the overlay over time.

2.4. New Modules/Algorithms

TVPP's object-oriented design includes generic interfaces for many modules. These interfaces provide flexibility to implement new features or algorithms. For instance, it is easy to create a new kind of message that will be exchanged between nodes since the send/receive methods receive a Message object as parameter and all messages inherit from Message. So, to create a new kind of message, one must only create a class that extends from Message, add a new entry at the group of message kinds, describe the structure of the new kind of message through an abstract method inherit from Message and introduce a method to handle that message.

Two other mechanisms which are easy to alter are the scheduler policy and the bootstrap peer selection algorithms. Both have been implemented as a strategy pattern. To extend these mechanisms, one must develop his algorithm as a extension of a strategy interface and patch the new strategy in with a few lines of code on headers and at the parameter handler.

Implementing a new peer performance metric and logging it is also simple. Since the log message is basically a wrapper, one must add the new metric at the log message, to the chain that constructs it and make sure that it will be unwrapped and written at the log file on the other side, in this case the bootstrap.

3. Usage

TVPP contains two applications, the bootstrap and the client. Both programs are Linux compatible, but can be compiled through Cygwin[6] to create a Windows executable. The programs can be run through command line.

[6]http://www.cygwin.com/

Clients can either be the channel source or a viewer. In order to start an experiment one must start a bootstrap, a channel source and any number of viewers. We recall that each program must have its own address, an ip-port pair. Usually our own experiments are conducted on PlanetLab[7] and we only attach one host to each node. More than one logical viewers can be located at the same physical host.Peers and bootstrap can be configurable to use specific ports.

As previously mentioned, the bootstrap stores data about each channel. Another parameter is channel id. The bootstrap can hold several channels, thus TVPP can perform parallel experiments. Since each channel creates a different overlay, results do not interfere with each other.

Finally, the most interesting parameters are those that can actually alter experimental results. Currently, one can fix the following set of parameters:

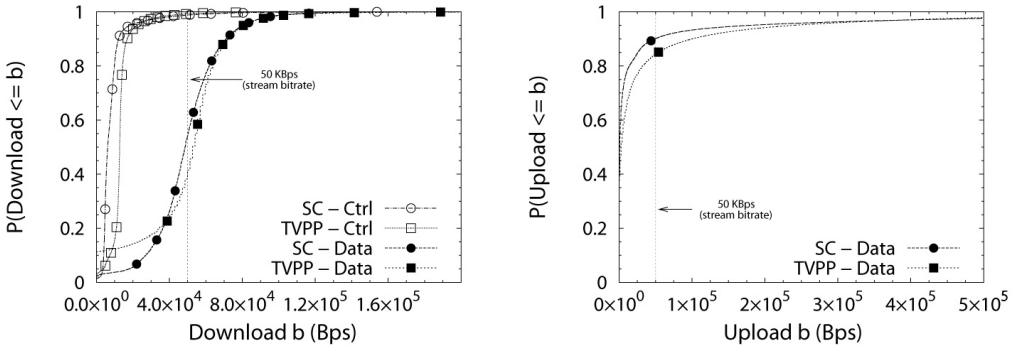
- mode, initially thought as a way to define if a peer will be a source or a viewer, this parameter can be used to create special kinds of peers, e.g. a free rider;
- buffer size, that affects buffer map message sizes and the peer capacity to hold chunks;
- maximum number of neighbors, relates with the overlay, more partners means more chunk sources, a more connected network, but it also increases buffer map messages throughput since they are periodically broadcasted to all neighbors;
- request limit, defines the amount of chunks that can be simultaneously requested;
- unresponsive partnership timeout, this timeout defines how many seconds a peer must wait before removing an unresponsive neighbor from its partners list;
- upload and download limits, these restrict TVPP to send or receive a maximum number of bytes using a leaky bucket;
- bootstrap's peer subset selection algorithm, the subset of peers returned by the bootstrap can be random, oriented by IP distance or RTT (round-trip time) between peers, but one can create his own selection strategy;
- partnership candidate selection algorithms, for connecting or disconnecting a neighbor, these algorithms may be the same as above since they are also peer selection strategies;
- chunk scheduler algorithm, another peer selection algorithm but with chunk transmission purposes.

Using this tool, we have already conducted some experiments. In order to illustrate its use, we have compared TVPP with SopCast(SC). We have setup over 500 PlanetLab nodes, have streamed a 100 minutes 400Kbps average bitrate video which looped continuously during our experiments and have captured data for 60 minutes in both SopCast and TVPP systems. TVPP allowed us to set neighborhood limits for server and clients at 10 and 50 peers, respectively, following characteristics observed on SopCast. Figure 1 presents a comparison between peer upload and download rates for both systems that shows one of many similarities between the systems. A point in those curves shows peers that have at least the indicated upload/download rate in a given second in time.

4. Related Work

In 2005, CoolStreaming [Zhang et al. 2005] gained academic notoriety as the first P2P live streaming system to publish information about its internal scheme. Its data-driven

[7]http://www.planet-lab.org/



(a) Download grouped by control and data packets

(b) Upload restricted to data packets

Figure 1. Cdf of download and upload rates of each peer in each second for both systems. Stream bitrate is ≈50KBps.

model has been extensively cited over the few years. Nevertheless, the system was never deployed as a testing platform and few measurements were done over it [Li et al. 2008].

Several measurement papers have been published that try to understand popular P2P live streaming systems such as PPLive and SopCast [Vieira et al. 2009, Horvath et al. 2008, Silverston et al. 2009, Ali et al. 2006, Tang et al. 2009] though few papers had proprietary data to work on, such as UUSee [Wu et al. 2007]. Most of the research published on popular systems rely on crawling the network with a subset of nodes and capturing network traffic, but that approach is inefficient while trying to answer some questions like those stated in Section 1. Especially, *what if* questions that propose abnormal situations.

The NAPA-WINE project [Abeni et al. 2010] is the closest effort toward building a P2P live streaming system with academic purposes in the last few years. Their work is focused at performance and system optimization, especially through network awareness, creating partnerships using insights from ALTO [Seedorf and Burger 2009]. They have designed a development toolkit and a set of libraries to facilitate the implementation and integration of experimental algorithms into their project. Their design also includes monitoring the latency and the available bandwidth between two peers, or the presence of Network Address Translation (NAT). Contrary to NAPA-WINE, which concentrates on measuring network behavior, TVPP focuses on plain experimentation and therefore can easily provide raw logs, receive new algorithms and extensions, and adapt to several different kinds of experiments, such as freeriding, or polluting clients.

5. Conclusions

In this paper we present TVPP, a P2P live streaming system that has been designed to simplify experiments and to assist researchers in better understanding the behavior of P2P-TV systems. TVPP works similarly to popular existing systems such as SopCast, but enables more efficient experimentation and data acquisition by eliminating the need of external network traffic analyzers, by allowing configuration of system parameters, and by providing a set of performance and overlay data. Moreover, TVPP can also provide additional information about system behavior, such as latency and chunk miss rate, that impact user experience but cannot be easily obtained in existing systems.

Figure 2. Screenshot of an experiment showing the bootstrap (top left), a channel source (bottom left) and two viewers with their players (middle and right).

This tool is already operational, supporting experiments and providing results in some of our studies. We expect to continue extending it to include new features. Meanwhile, we already envision a few tweaks that would make it even better and easier for researchers. Some of those are: embedding a configurable churn generator, using a xml-like file to configure experiments, altering parameters in execution time, and real-time graphical monitoring.

We believe that TVPP can provide an efficient alternative to P2P-TV experimentation, providing a more direct and straightforward way of evaluating new algorithms. In addition to that, we believe that TVPP can also be used as an efficient P2P live streaming system in real applications.

6. Demo Proposal

The demo presentation can be held with one or multiple computers. We expect to show a live capture with bootstrap, media source and other peers, deploying a real experiment on PlanetLab. Compile and run sequence on participants computers will also be stimulated. Figure 2 shows a sample experiment using 400+ nodes. In this experiment the source has sent media to the top viewer that redistributed it to the network. The bottom viewer has received the stream through the nodes in PlanetLab and therefore with a near 3 seconds latency. On each screen it can be observed output that show message exchange, expose peer lists and report about chunk IDs (what is the newer chunk, what is being requested, what have been received, what is being played). By default, we plan on starting an empty overlay with the chronometer video. Afterwards we will insert a single peer, then we shall remotely start hundreds of PlanetLab peers, and, finally, start another peer. This will show peers in two different layers of the network, one closer to the server and other farthest.

A TVPP stable version and documentation can be found at: *http://vod.dcc.ufmg.br/tvpp/*. Yet, newer versions are available through request.

References

Abeni, L., Bakay, A., Biazzini, M., Birke, R., Leonardi, E., Cigno, L., Kiraly, C., Mellia, M., Niccolini, S., Seedorf, J., et al. (2010). Network Friendly P2P-TV: The NapaWine Approach. In *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, pages 1–2. IEEE.

Ali, S., Mathur, A., and Zhang, H. (2006). Measurement of commercial peer-to-peer live video streaming. In *Proc. of Workshop in Recent Advances in Peer-to-Peer Streaming*. Citeseer.

Fodor, V. and Dan, G. (2007). Resilience in live peer-to-peer streaming [peer-to-peer multimedia streaming]. *Communications Magazine, IEEE*, 45(6):116–123.

Hei, X., Liu, Y., and Ross, K. (2008). IPTV over P2P streaming networks: the mesh-pull approach. *Communications Magazine, IEEE*, 46(2):86–92.

Horvath, A., Telek, M., Rossi, D., Veglia, P., Ciullo, D., Garcia, M., Leonardi, E., and Mellia, M. (2008). Dissecting PPLive, SopCast, TVAnts. *submitted to ACM Conext*.

Li, B., Xie, S., Qu, Y., Keung, G., Lin, C., Liu, J., and Zhang, X. (2008). Inside the new coolstreaming: Principles, measurements and performance implications. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 1031–1039. Ieee.

Seedorf, J. and Burger, E. (2009). Application-layer traffic optimization (ALTO) problem statement. *draft-marocco-alto-problem-statement-04 (work in progress)*.

Sentinelli, A., Marfia, G., Gerla, M., Kleinrock, L., and Tewari, S. (2007). Will IPTV ride the peer-to-peer stream?[Peer-to-Peer Multimedia Streaming]. *Communications Magazine, IEEE*, 45(6):86–92.

Silverston, T., Fourmaux, O., Botta, A., Dainotti, A., Pescapé, A., Ventre, G., and Salamatian, K. (2009). Traffic analysis of peer-to-peer IPTV communities. *Computer Networks*, 53(4):470–484.

Tang, S., Lu, Y., Hernández, J., Kuipers, F., and Van Mieghem, P. (2009). Topology dynamics in a P2PTV network. *NETWORKING 2009*, pages 326–337.

Vieira, A., Gomes, P., Rocha, M., Almeida, J., and Campos, S. (2009). A behaviour model of the SopCast users. In *Proceedings of the XV Brazilian Symposium on Multimedia and the Web*, pages 1–8. ACM.

Wu, C., Li, B., and Zhao, S. (2007). Magellan: Charting large-scale peer-to-peer live streaming topologies.

Zhang, X., Liu, J., Li, B., and Yum, T. (2005). CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming. In *proceedings of IEEE Infocom*, volume 3, pages 13–17. Citeseer.